Semi-automatic Mining of Semantic Descriptions of Processes in the Web

Julia Hoxha and Sudhir Agarwal
Insitute of Applied Informatics and Formal Description Methods (AIFB)
Karlsruhe Institute of Technology(KIT)
Karlsruhe,Germany
{*julia.hoxha,sudhir.agarwal*}*@kit.edu*

***Abstract*—Most of today's business processes are complex and consist of more than one party or single step procedures. In the Web, this is reflected by the existence of billions of Web sites, which may be regarded as complex processes, and on the other side only a few thousands of publicly available WSDL files that present single services. The availability of semantic descriptions of services and processes in the Web facilitates their discovery, as well as their composition into more complex workflows. It also facilitates the automatic execution of such workflows despite their heterogeneity. However, the deficit of semantic descriptions of Web processes deprives the users from using such sophisticated automatic techniques.**

The scope of our research is to fill this gap by providing semi-automatic techniques for the acquisition of a large number of semantic process descriptions on the (deep) Web. We model the data found in the online sources using ontologies, mine the process a user follows through the Web forms and generate a semantic description of this process. We present in this paper the implementation of our algorithms for the acquisition of process descriptions. We also provide a Web-based editor for manual annotation of new processes and refinement of the automatically-generated descriptions.

***Keywords*-web process; semantic process description; semantic web; process mining; deep web;**

I. Introduction

Most of the interesting processes today are complex, often requiring inputs from the user and providing outputs at multiple stages during execution. Furthermore, the execution path of a process often depends on the input values that the user provides. Today a large number of (business) processes are offered in the Web in form of services and websites, which we collectively refer to as *Web Processes*.

Consider for example a user performing a car rental search with conditions on pickup location, type of vehicle, winter tyres, booking with visa card, etc. In order to find the appropriate car rental page, a search engine needs to consider that (1) the terms used in the query may differ from those occurring on a car rental Website, (2) the information needed to answer the query may be scattered over multiple, even dynamically-generated pages of a car rental process (e.g. order and payment details appear often on separate pages).

While issue (1) has been dealt with by proposing ontologies for semantically modeling data on Web sites [1], there are no techniques available to capture and reason about the dynamics of Web processes. The scope of our research is to explore the content hidden in the Deep Web [2] behind subsequent HTML forms. We model the data found in the online sources using ontologies, mine the process a user follows through the forms and generate a semantic description of this process. Our goal is to develop techniques to obtain semantic descriptions of static and dynamic aspects of Web processes. We also aim to build a repository with a large number of such semantic descriptions [1].

Current search engines still work in a very document-centric way, since they ignore the flow of information among the pages entailing a Web-based process. There are only a few research works in the field with a similar goal to ours, to automatically discover and provide new semantic Web service descriptions. They either search in the literature [3] or in workflow definitions [4], such as in the bioinformatics domain, or explore the Web to find and create new semantic descriptions of services [5]. The drawback of the first two approaches is that they do not provide detailed semantic service descriptions or definition of their function. The work in [5] only covers the annotation of simple, atomic services with a single step procedure, but does not provide methods to discover more complex processes in the Web.

In this paper, we extend our previous work [6] and follow a semi-automatic mining approach for the acquisition of semantic process descriptions. We combine manual and automatic techniques because, on the one hand, pure manual approaches are less likely to scale to the size of the Web and, on the other hand, pure automatic techniques cannot always guarantee the correctness of the descriptions.

We show in Section II how we model the processes on Web sites and describe them semantically with a combination of process algebra and ontologies. In Section III, we present the automatic techniques developed for mining the semantic process descriptions. We also introduce our Web-based, collaborative environment for manual annotation of new processes and refinement of the automatically-generated descriptions. In Section IV, we present an implementation of our algorithms and discuss related work in Section V. Section VI concludes the paper with a summary of results and insights on future work.

[1]Such a repository can also be used by researchers to evaluate the performance and scalability of their techniques, e.g. semantic discovery, automatic composition, etc. on the descriptions of real processes.

II. Semantic Description of Web Processes

We semantically annotate the processes found in the Web using the formalism *suprimePDL* (suprime Process Description Language) [7]. In contrast to other existing semantic description formalisms, which either cover only the behavioral aspect of the processes or just the semantics of their input/output parameters, *suprimePDL* is able to cover both aspects and define interdependencies between them. In addition, it is a very expressive language and provides clear formal semantics.

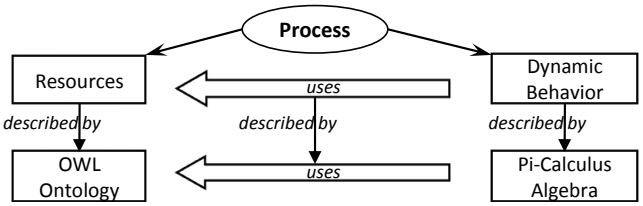

Figure 1. Process-oriented Modeling Formalism

Figure 1 shows the conceptual model of *suprimePDL* relevant for the work presented in this paper. The static data is described via ontologies using the decidable description logic $\mathcal{SHIQ}(\mathbf{D})$ [8], upon which is based OWL-DL, the decidable variant of the Web Ontology Language OWL. [2] The formalism proposes a process algebra, i.e. π-Calculus [9], to describe the dynamic behavior of processes. In the rest of this section, we give a short overview of *suprimePDL* and show how Web processes can be semantically described with it.

A. Overview of the suprime Process Description Language

In this section, we present the expressive language *suprimePDL* used for the semantic description of processes captured in the Web pages. The syntax of this formalism is given in Definition 1.

Definition 1: Syntax of Process Formalization

$$\begin{aligned} P \quad ::= \quad & \mathbf{0} \mid y[v_1 \ldots, v_n].P\prime \mid y\langle x_1 \ldots, x_n\rangle.P\prime \mid \\ & l(x_1, \ldots, x_n)(y_1, \ldots, y_m).P\prime \mid \\ & \omega?P_1{:}\, P_2 \mid P_1 \parallel P_2 \mid P_1 + P_2 \mid \ @A\{y_1, \ldots, y_n\} \end{aligned}$$

The elements in this definition have the following meaning. The *Null process* $\mathbf{0}$ denotes a process that is used as termination symbol in a process expression. The *Input process* $y[v_1, \ldots, v_n].P\prime$ is a process that takes inputs at port *y*, which is a communication channel, and binds them to names $v_1 \ldots, v_n$. The subsequent behavior of this process is defined in the process expression $P\prime$ that follows.

The *Output process* $y\langle x_1, \ldots, x_n\rangle.P\prime$ denotes a process that outputs the names $x_1 \ldots, x_n$ at port y. Its subsequent behavior is defined in the expression $P\prime$. The values $x_1, \ldots, x_n$ are instances in a domain ontology. The *Local process* $l(x_1, \ldots, x_n)(y_1, \ldots, y_m).P\prime$ denotes a process that performs the operation l with the arguments $x_1, \ldots, x_n$ and produces output $y_1, \ldots, y_m$. The input values $x_1, \ldots, x_n$ and output values $y_1, \ldots, y_m$ are instances in a domain ontology.

The *Deterministic Choice* $\omega?P_1{:}\, P_2$ is a process, whose behavior is determined by the condition ω. If ω is true, then its subsequent behavior is defined in the expression P_1 that follows. If false, it will be followed by expression P_2. The *Composition* $P_1 \parallel P_2$ consists of processes P_1 and P_2 acting in parallel. The *Summation* $P_1 + P_2$ denotes a non-deterministic choice of one of the alternatives P_1 or P_2. The selected choice determines the subsequent behavior of the process, defined in the selected expression.

Agent Identifier denotes a named process expression. In our context, we model as an agent identifier the URL where a Web page is located. $@A\{y_1 \ldots, y_n\}$ denotes the invocation of an agent identifier. More details about this formalism and its semantics may be found in [6], [7].

B. Modeling Content using Ontologies

The information we take into analysis for semantic description of websites is the textual page content, links and forms. Currently, our focus is on modeling links and forms, since they initially ensure the highest potential for navigating from one page to another via link execution (URL usage) and form submission.

Modeling Links. We consider three main conceptual elements in a link: URL base, variable names, and values. based on the typical convention of URL formation, we syntactically split the link into two basic parts: the string before "?" is saved as URL base, whereas the string following it is divided into variable names, which are view as ontology classes, and their respective values, viewed as instances of the class corresponding to the variable.

Modeling Forms. In order to successfully submit a form, we need to understand the semantic information it contains. Every form contains a set of field elements $E_1, E_2, \ldots, E_n$. We denote an element with $E_i(\mathit{label}, \mathit{name}, \mathit{domain})$, where $\mathit{label}(E_i)$ is its label found in the HTML page, $\mathit{name}(E_i)$ is the name of the variable bound to this element, and $\mathit{domain}(E_i)$ is the set of possible values this variable can have, e.g. the set of names in a dropdown menu, checkbox or radio button. Often this domain might not be defined in the form, e.g. in the case of textbox elements.

For each field element, we also store information we find in the structural units of the form. Therefore, we extract the layout of its different elements and model it not in just a flat representation, but a hierarchical structure containing significant semantic information. Figure 2 illustrates the hierarchical representation of the first page of a car rental website, which we consider as a running example in this paper.

[2] http://www.w3.org/2004/OWL/

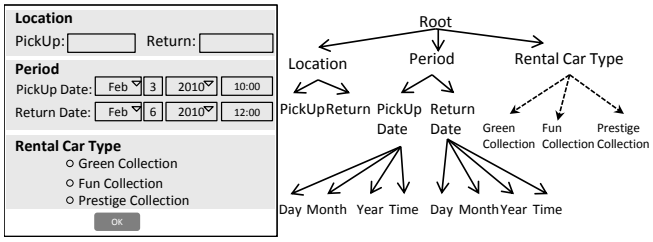

Figure 2. Hierarchical Representation of a Form

The first step of the semantic annotation is to capture the internal representation of each HTML form, apply normalization techniques (common case conversion, stop word removal, stemming) and model this information according to the Form Model Schema [10], [11] as illustrated in Table I.

Form Model Schema element	Description
Underlying source	URL base
Form template	Representation of form type. Each form template has (1) a layout (i.e. its graphical representation), (2) a title that identifies the HTML form and provides its general description, (3) an action method (GET or POST)
Form field	Element composed of (1) Label, (2) Variable name, (3) Variable type (text, submit, radio, select, etc.), (4) Domain of values (infinite or finite)
Structural unit	Logical group of closely related form fields, identified by a name (label)
Relationship	Relation between structural unit and the form fields it contains. It may also be an Inheritance
Association	Binary relation between form field label and variable name
Constraint	Rule that defines what data is valid for a given form field.A cardinality constraint specifies for a relationship the number of instances that a structural unit can participate in
Form instance	An occurrence of form type, when its template is filled in with data

Table I
FORM MODEL SCHEMA

This schema preserves the hierarchical representation of the form, using parent-child relations among its elements. A set of transformation rules, displayed in Table II, is then used to translate the form model schema to ontologies. In the ontology form, the elements are structured as classes and subclasses, e.g. "Pick-up Date" is a subclass of "Period", or classes and instances, e.g. "Green Collection" is an instance of "Rental Car Type".

C. Modeling Behaviour with Semantic Process Descriptions

Using the process description language *suprimePDL* we derive semantic description of processes, which capture the information flow and dynamics among the Web pages. We have defined a set of mappings (Table III) between the

Form Model Schema Element	maps to
Underlying source	Ontology
Form Title (part of Template)	class of Ontology
Structural unit	Class
Label of Form Field	Class
Variable name of Form Field	Instance
Association	equivalenceClass relation
Relationship	Property
Inheritance relationship of structural units	Inheritance of classes
Constraint	Axiom
Form Field Value	Instance

Table II
RULES OF TRANSFORMING FORM MODEL SCHEMA TO ONTOLOGY

Web Artefact	Element of the Process Description Language
URL	Agent Identifier
Web page	Message
Clicking a Link	Invocation of an Agent Identifier
Filling a Form	Input process
Submitting a Form	Invocation of an Agent Identifier
CGI script	Local operation

Table III
MAPPING BETWEEN WEB ARTIFACTS AND ELEMENTS OF THE FORMALISM

Web artifacts and the elements of *suprimePDL*. When a URL is accessed, the Web server generates a Web page and sends it to the client. A URL u with arguments $a_1, \ldots, a_n$ is modeled as an agent identifier $U(a_1, \ldots, a_n) \stackrel{def}{=} u(b_1, \ldots, b_m)(o_1, \ldots, o_l).c\langle o_1, \ldots, o_l \rangle.P$, where $\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_n\}$. The left-hand side of the expression defines the agentidentifier U with arguments $a_1, \ldots, a_n$, whereas the right-hand side describes the process of this agent, i.e. it includes a local operation u with input arguments $b_1, \ldots, b_m$ and outputs $o_1, \ldots, o_l$. These outputs denote the content of the HTML page retrieved after the operation. The page is then sent to the client in the subsequent output activity c. The outputs o_i may be links to other Web sites or submit buttons of the forms, each of them defining a possibility to navigate to the next page.

If an o_i is a link, it denotes the usage of a URL. In case of a form, the "action" can be either a link or the invocation of a method executed locally by the server (CGI script). The usage of a URL is equivalent to the invocation of an agent identifier, whereas a CGI script is equivalent to the invocation of a local operation.

Submission of a form binds the values entered in the form fields to the names of the corresponding fields. A form f with field names $v_1, \ldots, v_n$ can be seen as an input process $f[v_1, \ldots, v_n].P$, where P denotes the process representing the non-deterministic choice of all the submit buttons of the form f. That is, if the form f has submit buttons $b_1, \ldots, b_m$, then $P = \sum b_1, \ldots, b_m$. Each of the field names $v_1, \ldots, v_n$

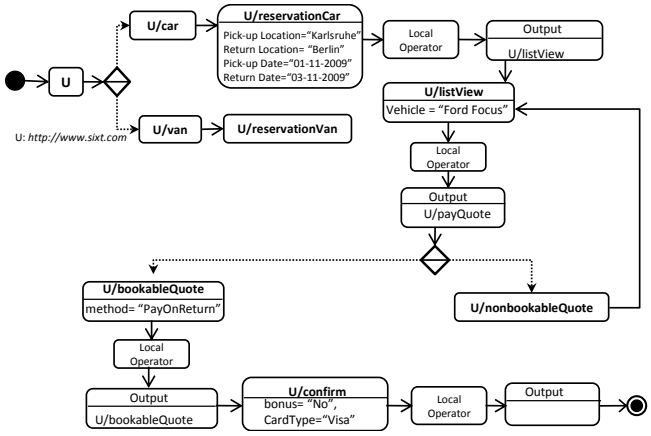

Figure 3. Graphic Representation of the Car Rental Process

is modeled as an ontology class, and the possible values of the field as respective instances of this class.

Example 1: Semantic description of a Car Rental Process on the Web (see Figure 3 for the graphical representation of the car rental process). Let U="http://www.sixt.com/"

$$
\begin{aligned}
U() &\stackrel{def}{=} @L_U(''U/car\text{“},''U/van\text{“}).c\langle''U/car\text{“},''U/van\text{“}\rangle.P_1\\
P_1() &= @L_{car}(U/resFormCar).c\langle U/resFormCar\rangle.P_{car} + \\
&\quad @L_{van}(U/resFormVan).c\langle U/resFormVan\rangle.P_{van}\\
P_{car}() &= p[PickLoc, RetLoc, PickDt, RetDt].@U/listView\\
&\quad (PickLoc, RetLoc, PickDt, RetDt)(h).c\langle h\rangle.P_{view}
\end{aligned}
$$

Initially, the URL U is defined as an agent identifier. Invocation of this URL is defined by the invocation of a local operation L_U, from which a page is generated containing two links "U/car“ and "U/van“. This page is displayed to the user (output denoted with $c\langle\rangle$). The behaviour that follows is further defined in the process expression P_1. Subsequently, a user may choose to click one of the links, defined as a non-deterministic choice. Based on this choice, the process either continues with P_{car} or P_{van}.

For each link invocation an operation is executed, then a page containing a *reservationForm* is given as output. The reservation form is then filled with values of fields labeled, respectively,"Pick-up Location“,"Return Location“,"Pick-up Date“,"Return Date“. In this way, we may continue describing the process followed by the selection of a vehicle, choosing the payment method ("PrepayOnline“, "PayOnReturn“), up to the final confirmation step.

III. Semi-automated Acquisition Approach

In this section, we explain the approach used for the acquisition of semantic process descriptions. It entails a combination of automatic and manual techniques. The scope of the automatic approach is to subsequently fetch pages on the Web via a crawler and semantically annotate the processes that we mine among these pages. The descriptions

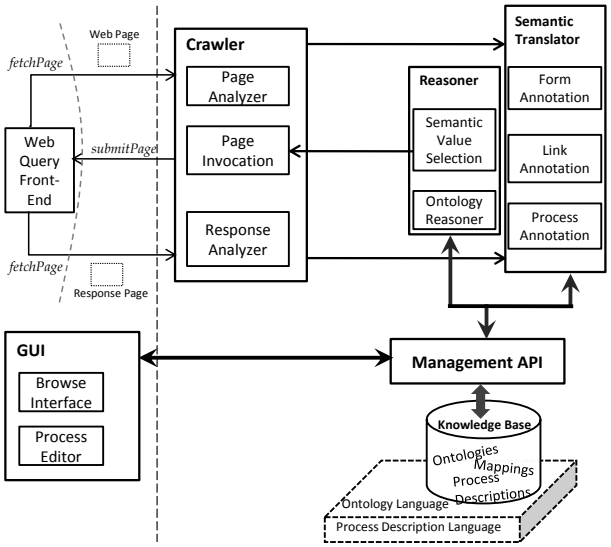

Figure 4. Semi-automated Acquisition Approach

of these processes are stored in a repository. The manual approach consists of a graphical editor, which supports *suprimePDL* process description language. The editor is used to annotate new processes or refine the automatically-generated process descriptions. Figure 4 gives an overview of the components and steps involved in our acquisition approach.

Crawler. The crawler is initially started with a predefined queue of URLs. The Web page located at each URL is fetched, its HTML content is analyzed and translated into a semantic representation using the Semantic Translator module.

Semantic Translator. This module translates the information found in links and forms into ontologies, using the approach introduced in section II-B. Variables names of the crawled hypertext links are saved as ontology classes, and their respective values as instances. This is done by the Link Annotation module. At the same time, for each extracted form with a set of field elements E_i, each field label *label*(E_i) is also stored as a class. The name *name*(E_i) of the field element and the values of its domain *domain*(E_i) are used to create ontological instances of this class. This is realized by the Form Annotation module (see Figure 4).

The module **Process Annotation** maps the artifacts of the Web page with the elements of our process description language as presented in Section II-C.

We extract from the content of the current Web page all its hypertext links and the URLs specified in the "action“ element of forms. These links are stored in the initial queue, so that each of them is crawled in the next iteration. This consists of the Page Invocation step, where a link from the queue is invoked or a form is submitted, and the page

received in its response is the Response Page.

Page Invocation. One of the most challenging task of the crawler for accomplishing page invocation is the form submission. In order to submit a form successfully, appropriate values should be chosen for its input field elements. We have to correctly build the URL action, which is then called for form submission. The selection of these values requires understanding of the pages and use of reasoning techniques to provide not only syntactically-correct, but also semantically-correct input parameters.

Semantic Value Selection. We solve the problem of input value selection for form submission using methods of discovering semantic relations among ontology classes. This is the case of a form with field element *E(label, name, domain)*, for which we need a set of input values that are not provided in the form itself, i.e. $domain(E) = \emptyset$. We use semantic relations of ontologies stored in the knowledge base, in order to find other classes that are equivalent to the ontology class *label(E)*. We use the instances of these equivalent classes as the set from which we choose an input value for field element E. For the reasoning tasks, this module uses the Ontology Reasoner.

Example 2: **Semantic Value Selection of Web Forms** Link *"http://www.wetter.com/germany/g03?town=Karlsruhe"* is previously crawled and semantically annotated in our system. Variable name "town" is stored as a class *Town* in the knowledge base, and "Karlsruhe" as instance of this class. In a later iteration, a form is extracted from another Web page, containing a textbox field with label "'Return City"', which is again stored as an ontology class *City*. In order to submit the form, an appropriate input value is to be chosen for this particular field. We use Semantic Relation Discovery to find equivalent classes or subclasses of *City*, in this case we find the class *Town* and use its instances (e.g. "Karlsruhe") as the set from which to choose the input value for the textbox *"'Return City"'*.

The inferred values are sent to the Value Selection module, which builds the final action URL of the form and passes it to Page Invocation. This module is responsible for the submission of the form using the provided URL and a protocol that is in accordance with the HTML specification *submission methods* "GET" or "POST".

Knowledge Base. There are three main conceptual parts of the repository, based on the types of stored resources: Ontology (classes and instances), Mapping and Process Description. Upon this knowledge base, we have built a Management API that is used to add new ontologies, mappings and process descriptions, as well as edit/delete existing ones.

Manual Acquisition. The manual part of our acquisition approach is illustrated in the lower section of Figure 4. Using the Browse Interface, a user may access and navigate the whole repository of process descriptions, which are displayed in a graphical representation. Via the Process Editor, new processes of the Web can be manually annotated, creating new semantic process descriptions that can be stored in the repository. Additionally, all the existing descriptions may be accessed and refined with the editor. Equivalence or subclass relations among ontology classes, which we use in the Semantic Value Selection module, may be defined manually. We are also investigating the deployment of techniques [12], [13] to achieve Ontology Mapping automatically within our acquisition approach.

A. Acquisition Algorithms

After having described the various components of our approach (see Figure 4), we further present how these modules interact with each-other and how the approach is formalized in the algorithms that we deploy.

The main method, which starts the crawler and implements the automatic acquisition of the descriptions, is displayed in Algorithm 1. We create an agent identifier for each unique URL q of the queue. The creation of the agent identifier is done, separately, for each link (Alg.1, Line 6) calling method *processLink* given in Algorithm 2, and for each form (Alg.1, Line 9) calling method *processForm* of Algorithm 3.

The content page p located at this URL is fetched using appropriate methods *invokeLinkFetchPageContent* and *submitFormFetchPageContent*, depending on the type of URL being processed. The links and forms found in p are extracted and added to the main queue (Line 21 of Algorithm 1 for links; Line 8 and 14 of Algorithm 3 for forms). In this way, they are iteratively processed in the next main loop of Algorithm 1.

For each of the extracted links and forms found on the page with URL q, we create an agent identifier and add it to set $\mathcal{N}$. This set is then used to expand the definition of the initial agent identifier of q, extending the process expression P with the Summation of identifiers in $\mathcal{N}$ (Alg.1, Line 29). This describes a process of non-deterministic choice, where the action URLs of forms and the links in the Web page p are the options from which a user may choose to continue in another page.

Algorithm 2 semantically annotates the link, extending accordingly the ontology with new classes and instances using method *mapLinkToOntology*. In this algorithm, a new agent identifier is created for the link (Alg. 2, Line 8). Referring to Figure 4, this algorithm is covered by the module Link Annotation.

Forms found in a Web page are separately processed in Algorithm 3. The action URL of a form is used as a link, which we store in the main queue and then invoke in order to submit this form. The HTML content of the form is extracted with method *getFormModelSchema* (Line 1) and translated to ontology classes (method *mapFormModelSchemaToOntology* in Line 2). Furthermore, a new agent identifier is created, whose definition (Line 7, 13) depends on the action

Input: Set Q of pairs(URL, type)
// type defines how to fetch content of page located at URL
Output: Global sets: Identifiers $\mathcal{A}$, Ontologies $\mathcal{O}$, local Operations $\mathcal{T}$

1 $\mathcal{A} := \emptyset$; $\mathcal{O} := \emptyset$; $\mathcal{T} := \emptyset$;
2 **while** $\mathcal{Q} \neq \emptyset$ **do**
3 **foreach** $q \in \mathcal{Q}$, *where* $q = (URL, type)$ **do**
4 $\mathcal{Q} := \mathcal{Q} - q$; *//dequeue* q
   // A_q - definition of Agent Identifier for q
5 **if** $type(q)$ = *"link"* **then**
6 A_q := processLink($q(URL)$); *// Alg. 2*
7 p := invokeLinkFetchPageContent($q(URL)$);
8 **else**
9 A_q := processForm($q(URL)$); *// Alg. 3*
10 p := submitFormFetchPageContent($q(URL)$);
11 **end**
12 $\mathcal{A} := \mathcal{A} \cup A_q$;
13 L_p := extractLinks(p);
14 F_p := extractForms(p);
15 $elements(p) := L_p \cup F_p$;
   // Refine definition A_q, use $elements(p)$ instead of h
16 A_q := refineDefinition($elements(p)$);
17 $\mathcal{N} := \emptyset$; *// Local set $\mathcal{N}$ of agent identifiers for p*
18 **foreach** $l \in L_p$ **do**
19 A_l := processLink(l); *// Algorithm 2*
20 $\mathcal{N} := \mathcal{N} \cup A_l$;
21 $\mathcal{Q} := \mathcal{Q} \cup (l$,"link"$)$;
22 $\mathcal{A} := \mathcal{A} \cup A_l$;
23 **end**
24 **foreach** $f \in F_p$ **do**
25 A_f := processForm(f); *// Algorithm 3*
26 $\mathcal{N} := \mathcal{N} \cup A_f$;
   // URL of f is queued in method processForm
27 $\mathcal{A} := \mathcal{A} \cup A_f$;
28 **end**
   // Refine process expression P in A_q
29 $P = \sum_{A_n \in \mathcal{N}} A_n$;
30 **end**
31 **end**

Algorithm 1: Automatic Acquisition of Semantic Process Descriptions

Input: URL l
Output: Agent Identifier A_l
// Denote with b the URL base (extract string before "?")
1 b := baseURL(l);
2 O_b := mapLinkToOntology(l);
// Let $\mathcal{P}$ be the set of pairs (name,value) of arguments in l
3 $\mathcal{P}$:= formSetArgumentPairs(l);
4 $names(l)$:= set of names in $\mathcal{P}$;
5 $values(l)$:= set of values in $\mathcal{P}$;
// Initialize a set of string h to $\emptyset$
// Create local operation T_b, add it to set of Operations $\mathcal{T}$
6 $T_b := L_b(names(l))(h)$;
7 $\mathcal{T} := \mathcal{T} \cup T_b$;
// Create agent identifier A_l
8 $A_l(names(l)) \stackrel{def}{=} @L_b(names(l))(h).c\langle h\rangle.P$;

Algorithm 2: Process Link

Input: Form f
Output: Agent Identifier A_f
1 $\mathcal{S}_f$= getFormModelSchema(f);
// Create ontology O_f, concepts and instances
2 O_f := mapFormModelSchemaToOntology($\mathcal{S}_f$);
// Denote with $names(f)$) set of field names
3 $names(f)$:= getFormFieldsNames($\mathcal{S}_f$);
// Let l denote action URL of f
4 l := buildActionURL(f);
// Let m denote action Method of f
5 **if** m = *"GET"* **then**
6 $names(l)$:= set of argument names in l;
   // Create agent identifier A_f,add its URL to main Queue
7 $A_f() \stackrel{def}{=} p[names(f)].@l\{names(f)/names(l)\}$;
8 $\mathcal{Q} := \mathcal{Q} \cup (l$,"link"$)$;
9 **else if** m = *"POST"* **then**
   // Initialize a set of string h to $\emptyset$
   // Create local operation type T_l, add it to set of Operations $\mathcal{T}$
10 $T_l := l(names(f))(h)$;
11 $\mathcal{T} := \mathcal{T} \cup T_l$;
   // Create agent identifier A_f,add its link to main Queue
12 $A_f() \stackrel{def}{=} p[names(f)].@l(names(f))(h).c\langle h\rangle.P$;
13 $\mathcal{Q} := \mathcal{Q} \cup (l$,"form"$)$;
14 **end**

Algorithm 3: Process Form Algorithm

method of the form. The technique realised here are covered by the module Form Annotation of Figure 4.

Special importance in Algorithm 3 has the method responsible for building the action URL of the form (Line 4), where we need to find approprite values for the input fields. This comprises our Semantic Value Selection module. We map each field label to an ontology class of our knowledge base, and choose the input values from the set of instances of this class. When no such class exists, we search for an equivalent class that matches our concept at hand (as explained in Example 2).

IV. Implementation

The presented automatic approach provides semantic descriptions of processes on the Web and annotation of their content with ontologies. We have implemented the algorithms and the techniques shown in Figure 4. We implemented a thread for each link, therefore the crawler browses all the given URLs simultaneously. Afterwards, it subsequently processes the links and forms found in the pages of the websites. As explained earlier, ontologies and process descriptions are generated with our acquisition algorithms. We have created a knowledge base, which contains OWL files for ontologies and XML files for the semantic descriptions of the processes. The ontologies, classes and instances are stored in this base using OWL API. We have

also implemented an API with Java classes and methods, which handle the management of process descriptions based on *suprimePDL* formalism. For reasoning tasks, we use Hermit OWL Reasoner [3].

Manual Annotation. The automatically-generated process descriptions may be syntactically or semantically incomplete. For example, our automatic acquisition technique cannot fetch password-protected Web pages. Our aim is to obtain very precise descriptions, in order to also support sophisticated use cases. To increase the quality of process descriptions, we have developed a graphical editor, which can be used by end-users to refine and complete the existing process descriptions, as well as manually add new ones.

The graphical editor is Web-based, built upon the open source version of the oryx editor [4], whose functionality can be extended via *stencil sets* to support new languages. We exploited this feature and defined stencil sets for our process description language and the ontology language as presented in Section II. The graphical editor is available online [5]. It also serves as an interface to show all the semantic process descriptions that we have automatically generated and stored in the repository using our techniques. We have further implemented a module that converts these descriptions into a predefined JSON format, which we import in the editor for graphical display. Via the editor the user may browse the repository, open a process description and edit it graphically. It enables modeling new process descriptions and ontologies, and save the refined or newly created process descriptions/ontologies in the repository.

***Suprime* Framework.** The presented acquisition approach is implemented as a part of our *Suprime* Framework (Intelligent Management and Usage of Processes and Services) [6], which entails a collection of semantic techniques for discovery, ranking, composition, execution and policy reasoning of processes. *Suprime* architecture is illustrated in Figure 5. The highlighted components are the modules of the acquisition approach that we have displayed in Figure 4. The goal of the acquisition part is the provision with semantic process descriptions, upon which we may then deploy intelligent tasks, e.g. composition in more complex workflows and their execution. These descriptions may be used for the elaboration of sophisticated tools and search strategies.

V. Related Work

Process mining is a field, which has undergone a body of research [14], [15], focusing on the discovery of explicit process models based on workflow event logs. However, these techniques require a set of structured logs with a predefined format, which is not possible to be provided in the context of Web pages that are extremely heterogeneous.

[3]http://hermit-reasoner.com
[4]http://oryx-editor.org/
[5]http://suprime.aifb.uni-karlsruhe.de/acquisition
[6]http://suprime.aifb.uni-karlsruhe.de/

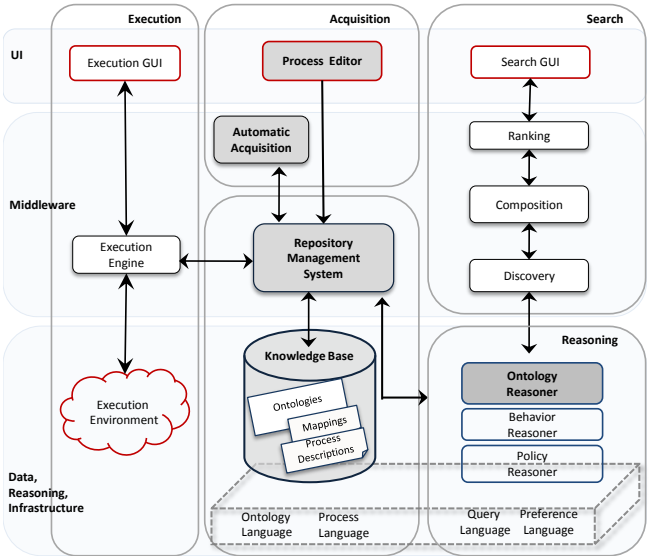

Figure 5. Architecture of *suprime* Framework

In [4] workflow definitions, which are compositions of atomic Web services, are explored to learn semantic types of input/output parameters of such services. In [3] textual information in the literature is used to infer descriptions of bionformatics services, providing a high level classification of services, but a limited description of their function. Neither [4] nor [3] can provide semantic descriptions for the services as detailed as our approach. The work in [5] explores the Web to find services and create semantic descriptions based on brute-force form submission. In our approach, we exploit the context information and incorporate reasoning based selection of values for the input parameters. Furthermore, we generate descriptions for more complex Web processes by submitting subsequent forms and considering the flow between them.

Research has been conducted for crawling the Web [16], [17], [18], in order to obtain the content of pages hidden behind HTML forms. Some of these approaches index the obtained information and use it to increase the coverage of the search engines. In difference to these approaches, our work is not restricted only to single pages, but it deals with a sequence of pages on the Web, in order to expressively describe the flow of information and the dependency among them. While the work in [18] focuses on discovering processes explicitly published on the Web, such as "'howto"' articles, our aim is to mine implicit processes among the pages, which are not explicitly published.

Similar to our research objective, seekda [7] is following the approach of crawling the Web and searching for publicly available Web services. The difference with our work is

[7]http://seekda.com/

that they focus only on the search of WSDL and RESTful services, while our context is broader. We crawl any page of the Web annotate its content and semantically describe as a process the flow among the pages.

VI. Conclusion and Future Work

The work presented in this paper is mainly motivated by the unavailability of semantic descriptions of dynamics of Web processes. We introduce an approach to fill this gap, mining subsequent pages of the deep Web and semantically describing the implicit processes captured in these pages using our *suprimePDL* formalism.

Our main contribution in this paper is a semi-automatic technique to build a repository of semantic descriptions of Web sites, which offers a view of the Web not only from the content perspective, but also from a process-oriented perspective. The automatic part of our approach produces descriptions of the content and flow among Web pages. The generated descriptions can be refined and new processes can be annotated via a Web-based process description editor. This is another contribution of our work that helps recover the scarcity of resources for manual annotation.

We are working on elaborating our techniques to find the semantics of the function that maps the input parameters of a form to the output variables, which we find upon form submission on the subsequent page. Since our goal in this paper is to introduce the process-oriented view of the Web and the acquisition approach, we are planning in the future to deploy techniques for the evaluation of the process descriptions. Our strategy is to produce a large number of descriptions randomly and create dynamic Web sites out of them, which will serve as our test benchmark. We plan to use our acquisition approach on these sites and get the semantic process descriptions, which we will evaluate towards the randomly-produced descriptions using graph comparison techniques.

References

[1] Handschuh, S., Staab, S.: Annotation for the Semantic Web. IOS Press (2003)

[2] Bergman, M.: The Deep Web: Surfacing Hidden Value. Journal of Electronic Publishing **7** (2001)

[3] Afzal, H., Stevens, R., Nenadic, G.: Mining semantic descriptions of bioinformatics web services from the literature. In: ESWC'09: Proceedings of the 6th European Semantic Web Conference, Springer-Verlag (2009)

[4] Belhajjame, K., Embury, S.M., Paton, N.W., Stevens, R., Goble, C.A.: Automatic annotation of web services based on workflow definitions. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, Springer-Verlag (2006) 116–129

[5] Ambite, J.L., Darbha, S., Goel, A., Knoblock, C.A., Lerman, K., Parundekar, R., Russ, T.: Automatically constructing semantic web services from online sources. In: ISWC '09: Proceedings of the 8th International Semantic Web Conference, Berlin, Heidelberg, Springer-Verlag (2009) 17–32

[6] Agarwal, S.: Semi-Automatic Acquisition of Semantic Descriptions of Web Sites. In: Proceedings of The Third International Conference on Advances in Semantic Processing, Malta, IEEE (2009)

[7] Agarwal, S., Rudolph, S., Abecker, A.: Semantic Description of Distributed Business Processes. In: AAAI Spring Symposium - AI Meets Business Rules and Process Management, Stanford, USA (2008)

[8] Horrocks, I., Sattler, U., Tobies, S.: Practical Reasoning for Expressive Description Logics. In: Proc. of LPAR'99. Volume 1705., Springer-Verlag (1999) 161–180

[9] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I+II. Journal of Information and Computation (September 1992) 1–87

[10] Astrova, I.: Reverse Engineering of Relational Databases to Ontologies. In: ESWS. Volume 3053. (2004) 327–341

[11] Benslimane, S.M., Malki, M., Rahmouni, M.K., Benslimane, D.: Extracting Personalised Ontology from Data-Intensive Web Application: an HTML Forms-Based Reverse Engineering Approach. Informatica **18**(4) (2007) 511–534

[12] Sabou, M., Motta, E.: SCARLET: Semantic Relation Discovery by Harvesting Online Ontologies. The Semantic Web: Research and Applications (2008) 854–858

[13] Voelker, J., Haase, P., Hitzler, P.: Learning Expressive Ontologies. In: Ontology Learning and Population: Bridging the Gap between Text and Knowledge. Volume 167. IOS Press (2008) 45–69

[14] Rembert, A.J.: Comprehensive workflow mining. In: Proceedings of 44-th ACM-SE, USA, ACM (2006) 222–227

[15] Van der Aalst, W., Vandongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, a.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering **47** (2003) 237–267

[16] Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. Proceedings of the International Conference on Very Large Data Bases (2001) 129138

[17] Madhavan, J., Ko, D., Kot, b., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's Deep Web crawl. Proceedings of the VLDB Endowment archive **1** (2008) 12411252

[18] Liu, Y., Agah, A.: Crawling and Extracting Process Data from the Web. Work (2009) 545–552